Εισαγωγή στον Προγραμματισμό

Εισαγωγική ενότητα (25%) : Βασικές έννοιες και τομείς της Επιστήμης Υπολογιστών 4η ομιλία

Παναγιώτης Τζουνάκης

Χειμώνας 2026



Τμήμα Μαθηματικών
Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης



Γλώσσες προγραμματισμού

1. Ορισμός της έννοιας «πρόγραμμα», ταξινόμηση και σύγκριση γλωσσών προγραμματισμού
2. Κυριότερα είδη προγραμματισμού και βασικά χαρακτηριστικά των αντίστοιχων τεχνικών
3. Πλεονεκτήματα του δομημένου προγραμματισμού
4. Η διαδικασία εκτέλεσης ενός προγράμματος
5. Βασικά προγράμματα σε ένα προγραμματιστικό περιβάλλον





Γλώσσες προγραμματισμού

Μέχρι τώρα είδαμε πως αλγόριθμοι χειρίζονται δεδομένα (οργανωμένα σε διάφορες δομές) για την επίλυση προβλημάτων

Πρόγραμμα είναι το σύνολο των εντολών-περιγραφών που εισάγονται στον υπολογιστή για να υλοποιηθεί ο αλγόριθμος που επεξεργάζεται τα δεδομένα και έτσι επιλύει ένα πρόβλημα.





Γλώσσες προγραμματισμού

Λεξικό της κοινής νεοελληνικής (Τριανταφυλλίδη) :

γλώσσα *: ...**2.** (μτφ.) οποιοδήποτε άλλο μέσο, εκτός από το λόγο, που βοηθάει στη συνεννόηση: ... Γλώσσες των ηλεκτρονικών υπολογιστών, μια καθορισμένη συλλογή χαρακτήρων και κανόνων, η οποία χρησιμοποιείται για το σχηματισμό συμβόλων, λέξεων κτλ., καθώς και οι κανόνες για το συνδυασμό τους σε κατανοητές για τους υπολογιστές έννοιες.*

Δηλαδή, οι χαρακτήρες (του αλφαβήτου) συνδυάζονται σε λέξεις και σύμβολα. Η διάρθρωση – δομή των λέξεων και συμβόλων ορίζει το νόημα των εντολών που πρέπει να εκτελέσει ο υπολογιστής κατά τη λειτουργία του.





Γλώσσες προγραμματισμού

Οι εντολές μπορεί να διατυπωθούν σε πολλές διαφορετικές τυπικές περιγραφές (γλώσσες προγραμματισμού) με ποικιλία μορφών (συντακτικό) και νοηματοδότηση (σημασιολογία – semantics).

Το πρόγραμμα πρέπει να διατυπώνει τον αλγόριθμο με τρόπο σαφή (μη διφορούμενο – unambigious), σε κατάλληλο επίπεδο λεπτομέρειας έτσι ώστε, το αντίστοιχο λογισμικό/υλικό του υπολογιστή που «βλέπει» το πρόγραμμα, να μπορεί να υλοποιήσει τον αλγόριθμο εκτελώντας (τρέχοντας) το πρόγραμμα.





Περί Γλωσσών Υπολογιστή

- Construction language, γενική, περιλαμβάνει configuration, toolkit, programming languages
  - Command language, a language used to control the tasks of the computer itself, e.g. as starting other programs
  - Configuration language, a language used to write configuration files
  - Programming language, a formal language για τη μεταβίβαση εντολών σε μηχανή, ειδικότερα υπολογιστή
    - Assembly language, κοντά στο επίπεδο της γλώσσας μηχανής, χρησιμοποιεί μνημονικές εκφράσεις για διευκόλυνση του προγραμματιστή
    - Scripting language, a programming language for a special run-time environment that automates the execution of tasks; the tasks could alternatively be executed one-by-one by a human operator
- Machine language (machine code) : σύνολο εντολών εκτελέσιμο κατευθείαν από το υλικό (hardware) της CPU
- Markup language, a grammar for annotating a document in a way that is syntactically distinguishable from the text, such as HTML
- …
- Query language, a language used to make queries in databases and information systems
- … αναλυτικότερα : https://en.wikipedia.org/wiki/Computer_language

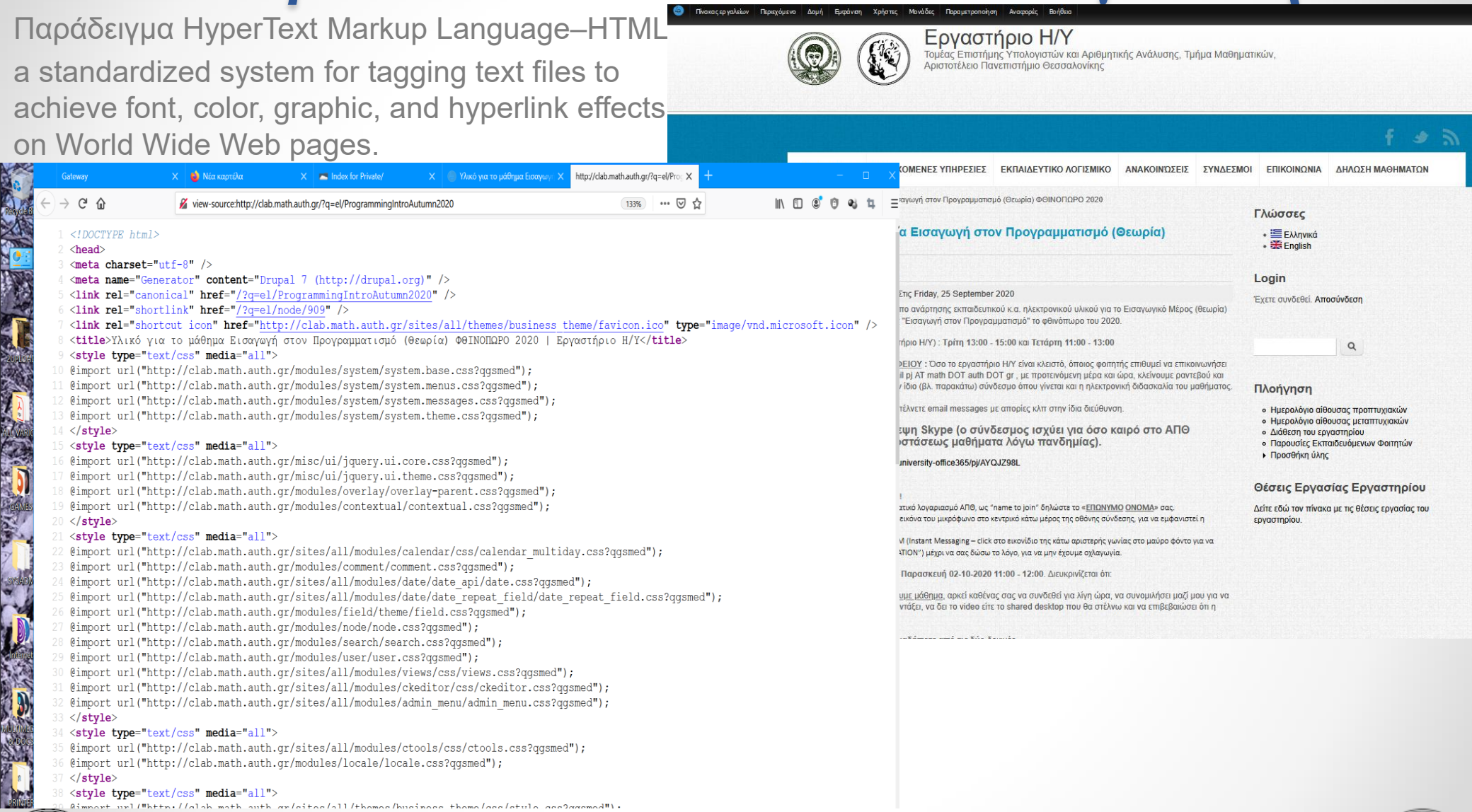




Περί Γλωσσών Υπολογιστή

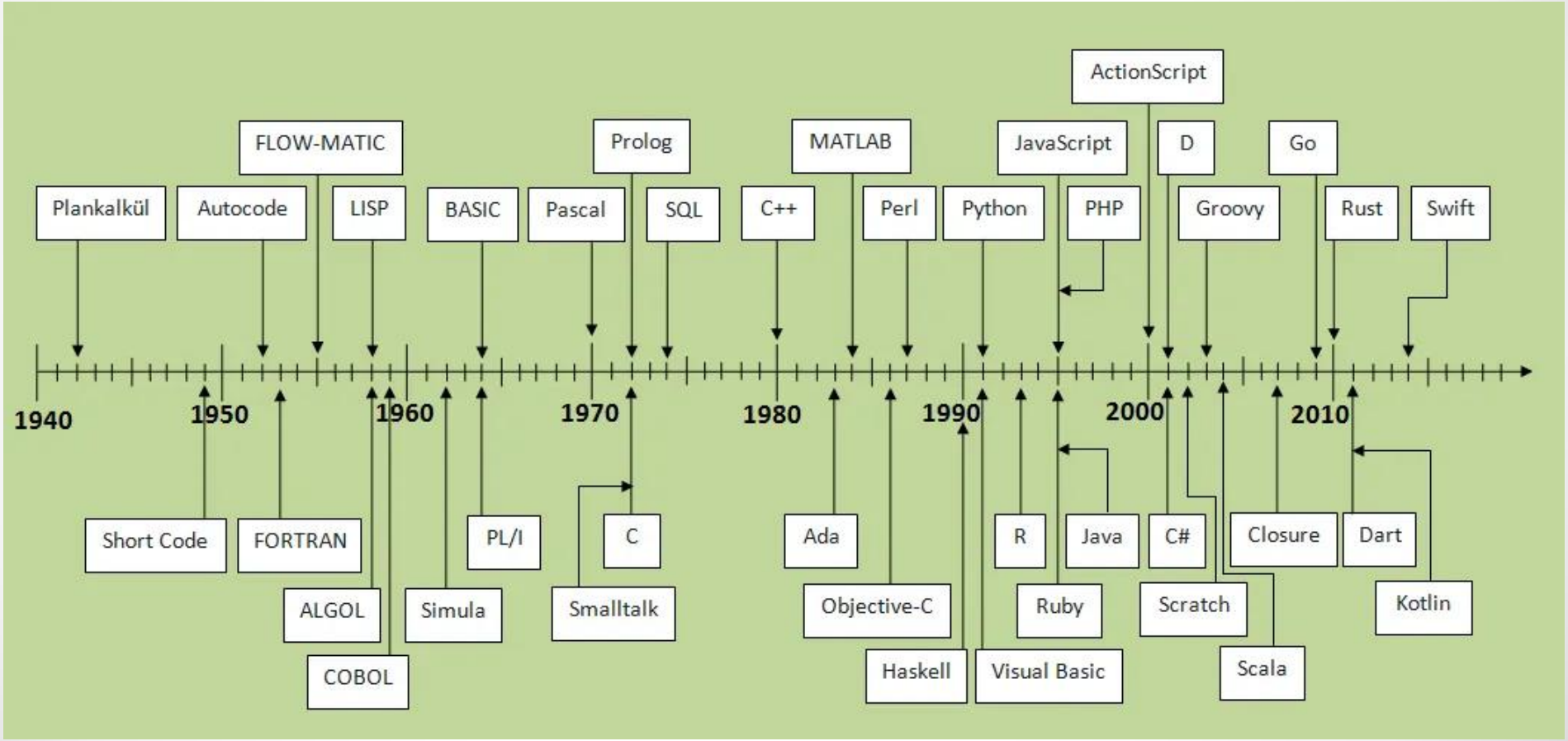
Παράδειγμα HyperText Markup Language–HTML

a standardized system for tagging text files to achieve font, color, graphic, and hyperlink effects on World Wide Web pages.



Ιστορικά στοιχεία και εξέλιξη

- https://en.wikipedia.org/wiki/Timeline_of_programming_languages
- https://www.i-programmer.info/history/computer-languages/2795-programming-language-infographic.html



https://javaconceptoftheday.com/history-of-programming-languages/

Το προγραμματιστικό υπόδειγμα

Ορισμός: ένα «πρότυπο ανάπτυξης/καθορισμένη μεθοδολογία» για την ανάπτυξη της δομής και των στοιχείων του προγράμματος.

- Προστακτικός (imperative) προγραμματισμός π.χ. FORTRAN, C, C++, Perl
- Δηλωτικός (declarative) π.χ. yacc parser generator, Make build specification
- Συναρτησιακός (functional) π.χ. Lisp, F#
- Αντικειμενοστραφής (objective) π.χ. C++
- Λογικός (logic) π.χ. Prolog
- Παράλληλος (parallel) *μόνο για προβλήματα που επιδέχονται «παραλληλοποίηση»
- Ο δομημένος προγραμματισμός:
  - Ιεραρχική σχεδίαση
  - Τμηματικός προγραμματισμός
  - Αποφυγή της «περιττής» εντολής GOTO (γνωστή και ως ”jump”)





Το προγραμματιστικό υπόδειγμα

Το «μαύρο πρόβατο» : η εντολή GOTO





Το προγραμματιστικό υπόδειγμα

Παράδειγμα της εντολής GOTO

```
/* To read and print the number, if number
is positive only*/
#include <stdio.h>
int main()
{
   int number;

   printf("Enter an integer number: ");
   scanf("%d",&number);

   if(number<=0)
      goto end;
   printf("Number is : %d", number);

end:
   printf("Bye Bye !!!");
   return 0;
}
```

Output

```
First run:
Enter an integer number: 123
Number is : 123
Bye Bye !!!

Second run:
Enter an integer number: 0
Bye Bye !!!
```





Επίπεδα γλωσσών προγραμματισμού

- Μηχανής
- Χαμηλό (συμβολικές – assembly γλώσσες)
- Υψηλό
  - Τρόπος έκφρασης κοντά στον ανθρώπινο
  - Ανεξαρτησία από τον τύπο (υλικό, λειτουργικό σύστημα, κ.α.) του Η/Υ. Τα προγράμματα είναι φορητά (portable) δηλ. μπορούν να μεταφερθούν σε άλλες υπολογιστικές πλατφόρμες με καθόλου ή ελάχιστες αλλαγές.
  - Ευκολία εκμάθησης
  - Ευκολότερη διόρθωση λαθών και συντήρηση προγραμμάτων σε σχέση με χαμηλότερα επίπεδα

  Αύξηση παραγωγικότητας με ελάττωση χρόνου και κόστους παραγωγής προγραμμάτων τα οποία καλύπτουν ευρύτερη γκάμα υπολογιστών
- Γλώσσες 4ης γενιάς (https://en.wikipedia.org/wiki/Fourth-generation_programming_language)
  - Περισσότερο αφηρημένες, ευέλικτες, συμβολικές





Καθοριστικά στοιχεία γλώσσας προγραμματισμού

- Αλφάβητο
- Λεξιλόγιο
- Γραμματική
  - Μορφολογία
  - Συντακτικό
- Σημασιολογία





Συντακτικό

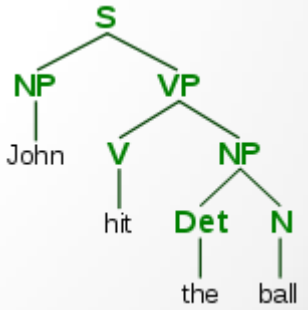
https://en.wikipedia.org/wiki/Syntax_(programming_languages)

Επίπεδα

- Λέξεις – λεξικολογικό (lexical) επίπεδο, καθορίζει πως οι χαρακτήρες σχηματίζουν tokens (λεξικογραφικές μονάδες)
- Φράσεις – επίπεδο γραμματικής, με τη στενή έννοια, καθορίζει πως tokens σχηματίζουν φράσεις
- Περιβάλλον (συμφραζόμενα) – καθορίζουν σε τι αναφέρονται τα αντικείμενα ή ονόματα μεταβλητών, αν οι τύποι είναι έγκυροι, κλπ.

Δύο στάδια συντακτικής επεξεργασίας

- Λεξικογραφική ανάλυση (χαρακτήρες -> tokens
- Συντακτική ανάλυση (parsing)



A simple parse tree

Σημασιολογία

- https://en.wikipedia.org/wiki/Semantics_(computer_science)
- https://el.wikipedia.org/wiki/Τυπική_σημασιολογία)των_γλωσσών_προγραμματισμού

Κύριες κλάσεις semantics

- **Δηλωτική σημασιολογία** (**Denotational semantics**)
  - Αφηρημένες έννοιες (concepts) τις οποίες περιγράφει η φράση της γλώσσας (functional languages)
- **Λειτουργική σημασιολογία** (**Operational semantics**)
  - Διερμηνεία των εκτελούμενων πράξεων-μετασχηματισμών που περιγράφει η φράση (π.χ. lambda calculus)
- **Αξιωματική σημασιολογία** (**Axiomatic semantics**)
  - Προτάσεις/αξιώματα μιας λογικής που ισχύουν για μια φράση (π.χ. Hoare logic)

Οντολογία (Ontology) : Στην επιστήμη της πληροφορίας, μια οντολογία περιλαμβάνει μια αναπαράσταση, μια επίσημη ονομασία και ορισμούς των κατηγοριών, των ιδιοτήτων και των σχέσεων μεταξύ των εννοιών, των δεδομένων ή των οντοτήτων που αφορούν έναν, πολλούς ή όλους τους τομείς τους οποίους συζητούμε. https://en.wikipedia.org/wiki/Ontology_(information_science)

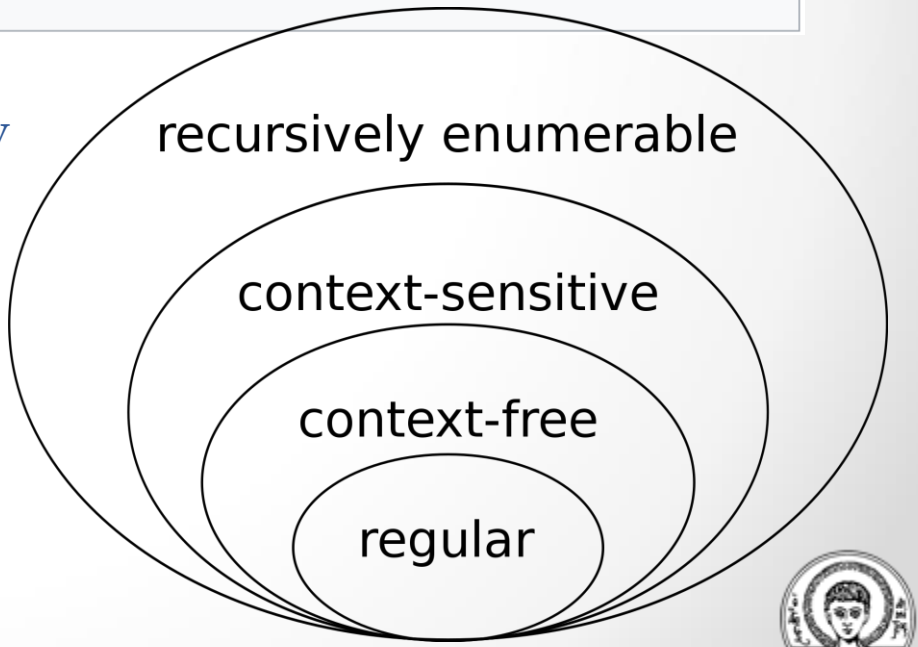
Ιεραρχία Chomsky

Grammar	Languages	Automaton	Production rules (constraints)*	Examples[3]
Type-0	Recursively enumerable	Turing machine	$\gamma \to \alpha$ (no constraints)	$L = \{w \mid w \text{ describes a terminating Turing machine}\}$
Type-1	Context-sensitive	Linear-bounded non-deterministic Turing machine	$\alpha A \beta \to \alpha \gamma \beta$	$L = \{a^n b^n c^n \mid n > 0\}$
Type-2	Context-free	Non-deterministic pushdown automaton	$A \to \alpha$	$L = \{a^n b^n \mid n > 0\}$
Type-3	Regular	Finite state automaton	$A \to \mathrm{a}$ and $A \to \mathrm{a}B$	$L = \{a^n \mid n \geq 0\}$

* Meaning of symbols:

- a = terminal
- A, B = non-terminal
- α, β, γ, δ = string of terminals and/or non-terminals
  - α, β, δ = maybe empty
  - γ = never empty

https://en.wikipedia.org/wiki/Chomsky_hierarchy



Παρένθεση: Δείγμα υλικού πάνω στα θεωρητικά θεμέλια της επιστήμης υπολογιστών



Μόνο για λάτρεις της μαθηματικής θεωρίας πίσω από τους υπολογιστές!

"On Computable Numbers, with an Application to the Entscheidungsproblem" by A. M. Turing, 1937 (https://doi.org/10.1112/plms/s2-42.1.230)

https://en.wikipedia.org/wiki/Turing_machine

https://en.wikipedia.org/wiki/Entscheidungsproblem

https://en.wikipedia.org/wiki/Church-Turing_thesis

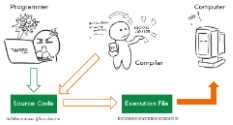
https://plato.stanford.edu/entries/turing-machine/

https://cacm.acm.org/magazines/2020/1/241712-von-neumann-thought-turings-universal-machine-was-simple-and-neat/fulltext

Σημείωση: Ηρεμήστε, δεν αποτελεί ύλη για τις εξετάσεις!!!...

Στάδια ανάπτυξης λογισμικού

Η καθημερινότητα του προγραμματιστή:

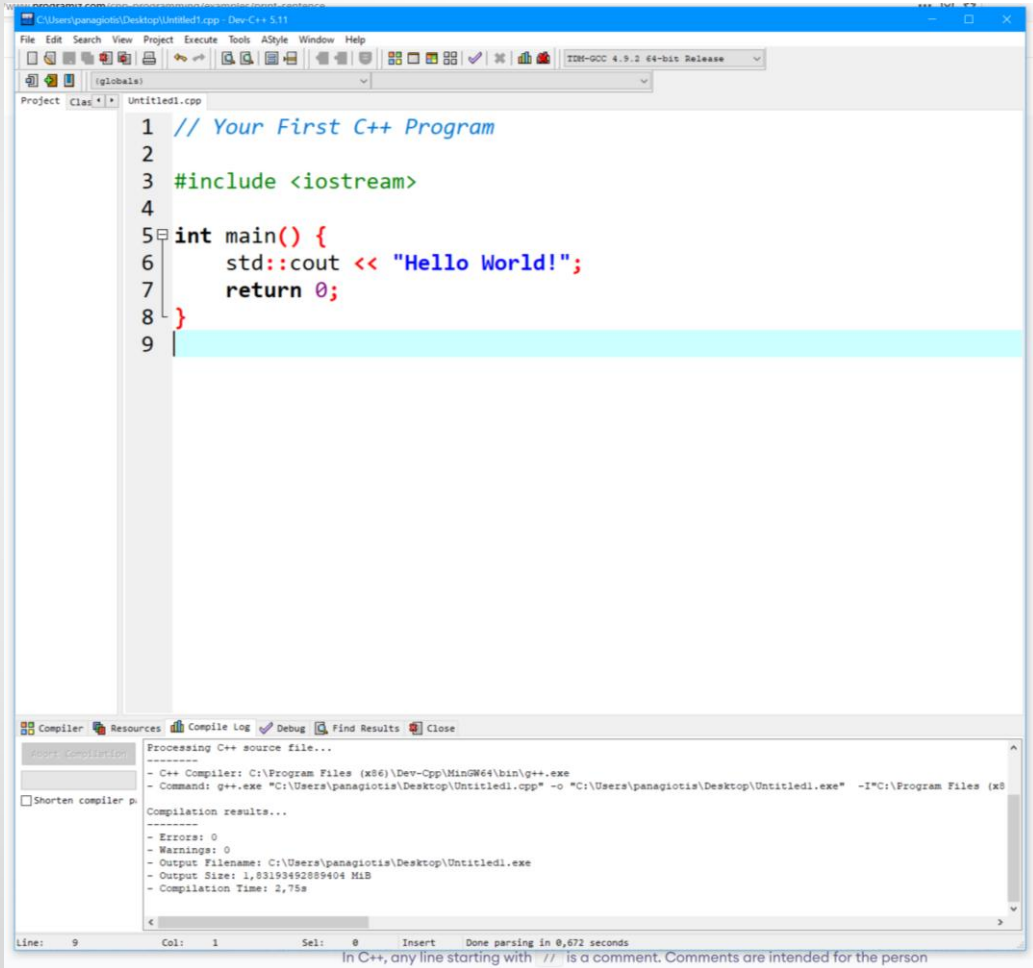
- Συγγραφή (edit code) 
- Μεταγλώττιση (compile) 
- Εκσφαλμάτωση (debug) 

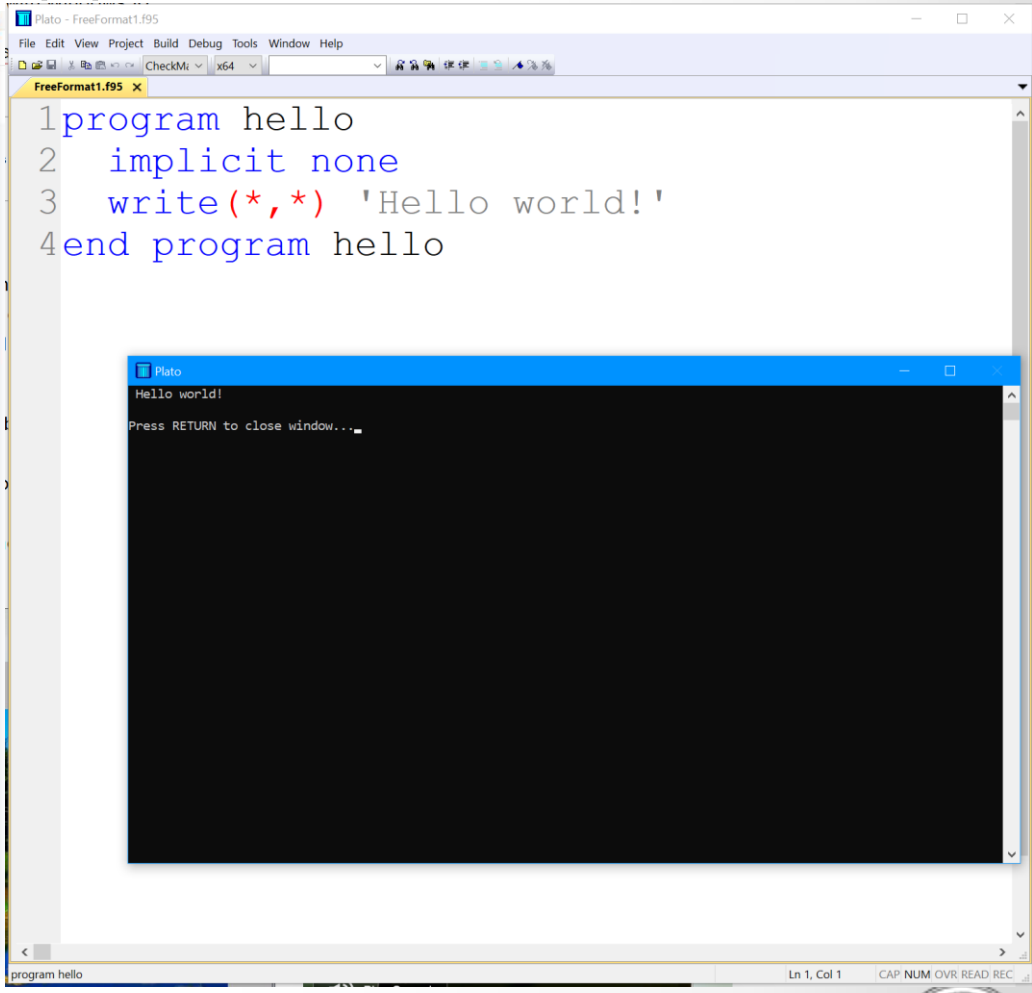
(αυτοδύναμα) τμήματα του κώδικα (ή και ολόκληρα προγράμματα) μπορεί να αντιμετωπίζονται ως «Μαύρο κουτί» : Εξετάζουμε τις εισόδους και εξόδους τους, θεωρώντας δεδομένη τη λειτουργία που (υποτίθεται ότι) επιτελούν.



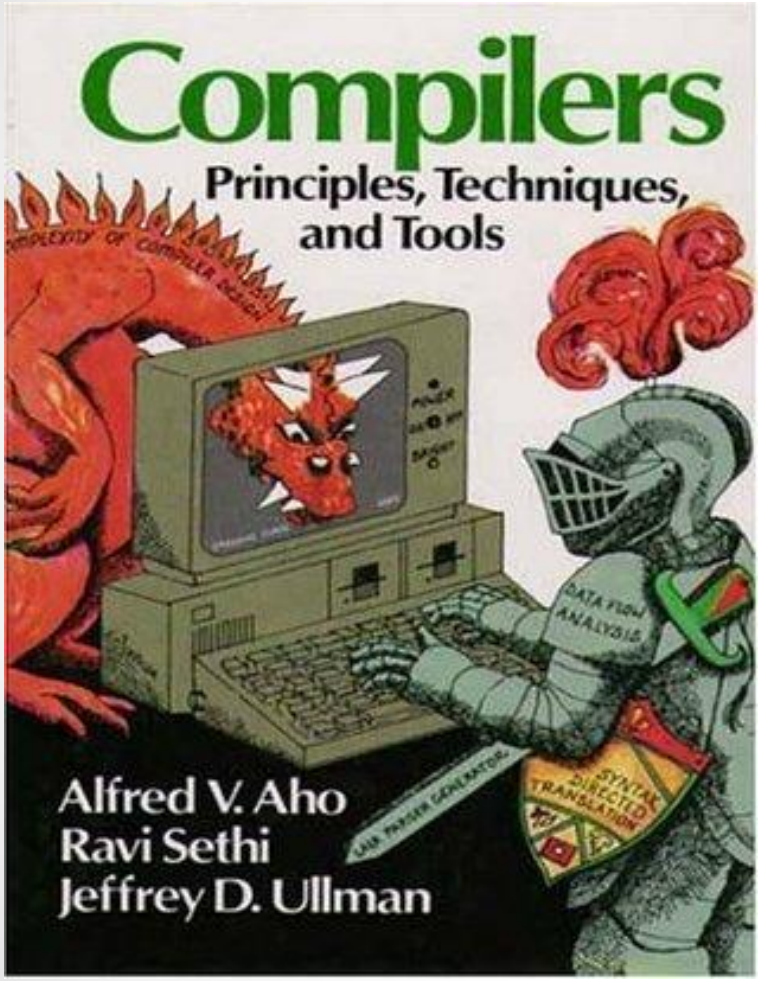
Συγγραφή

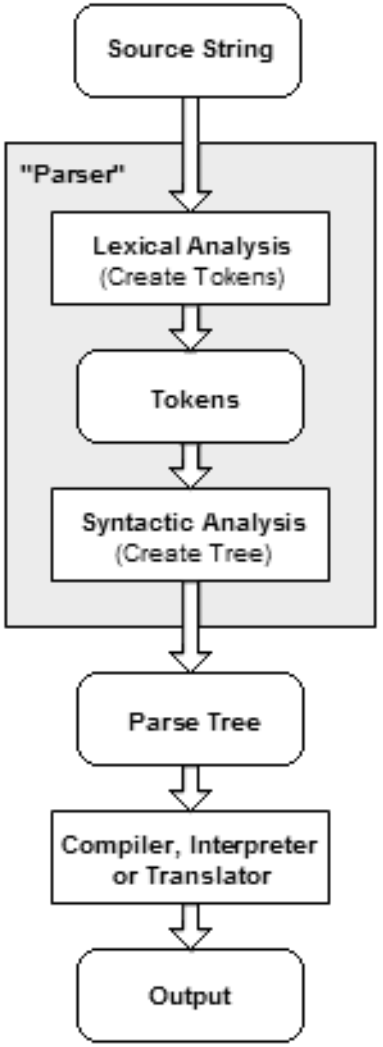
Μπορεί να χρησιμοποιούμε από απλό επεξεργαστή κειμένου, μέχρι ολοκληρωμένο προγραμματιστικό περιβάλλον…

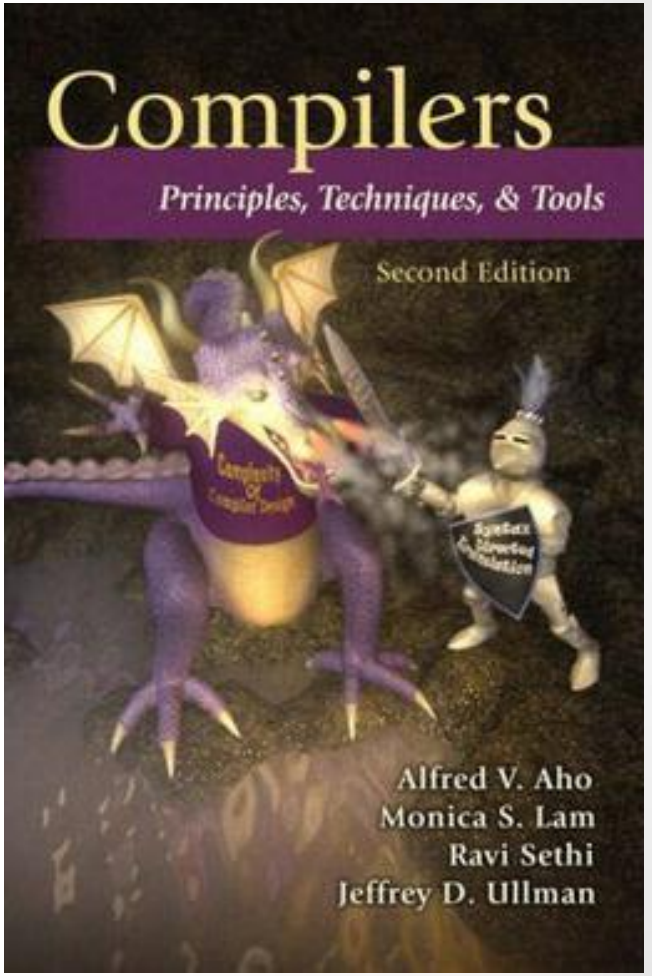




Μεταγλώττιση

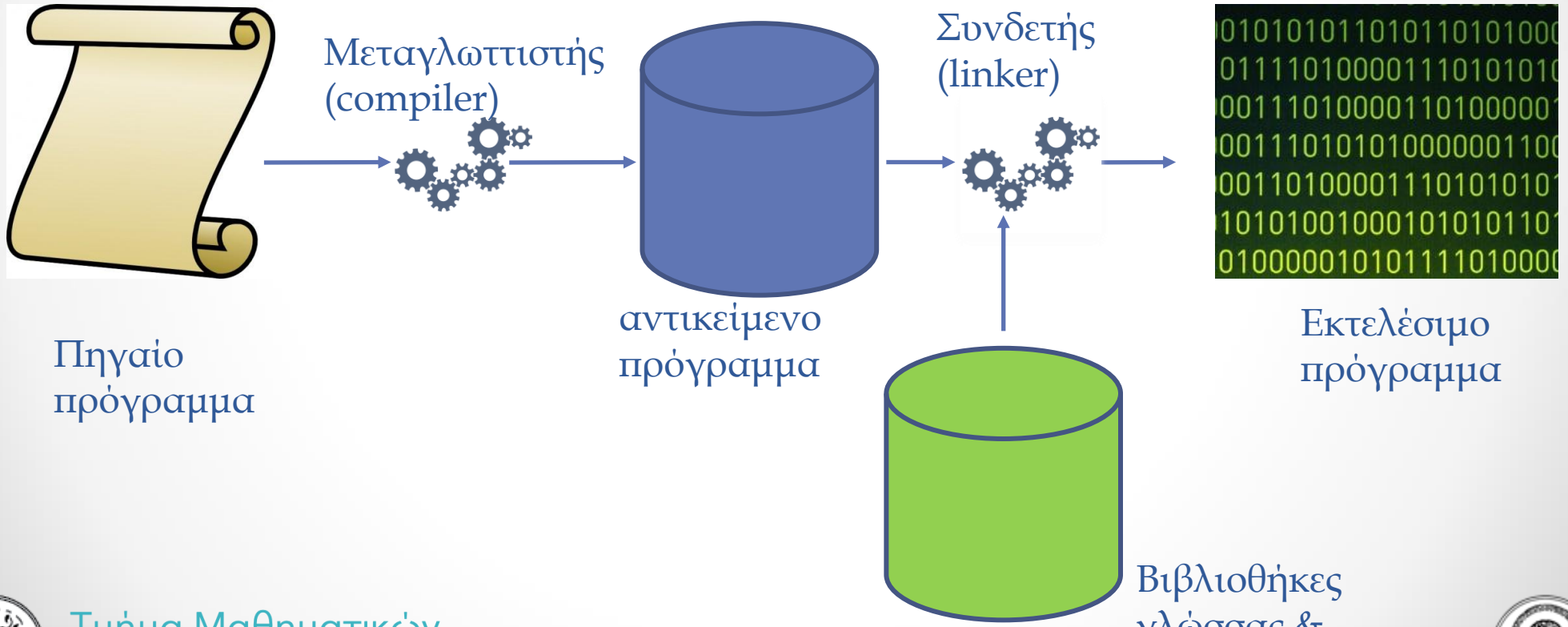






the "**Dragon Book**"

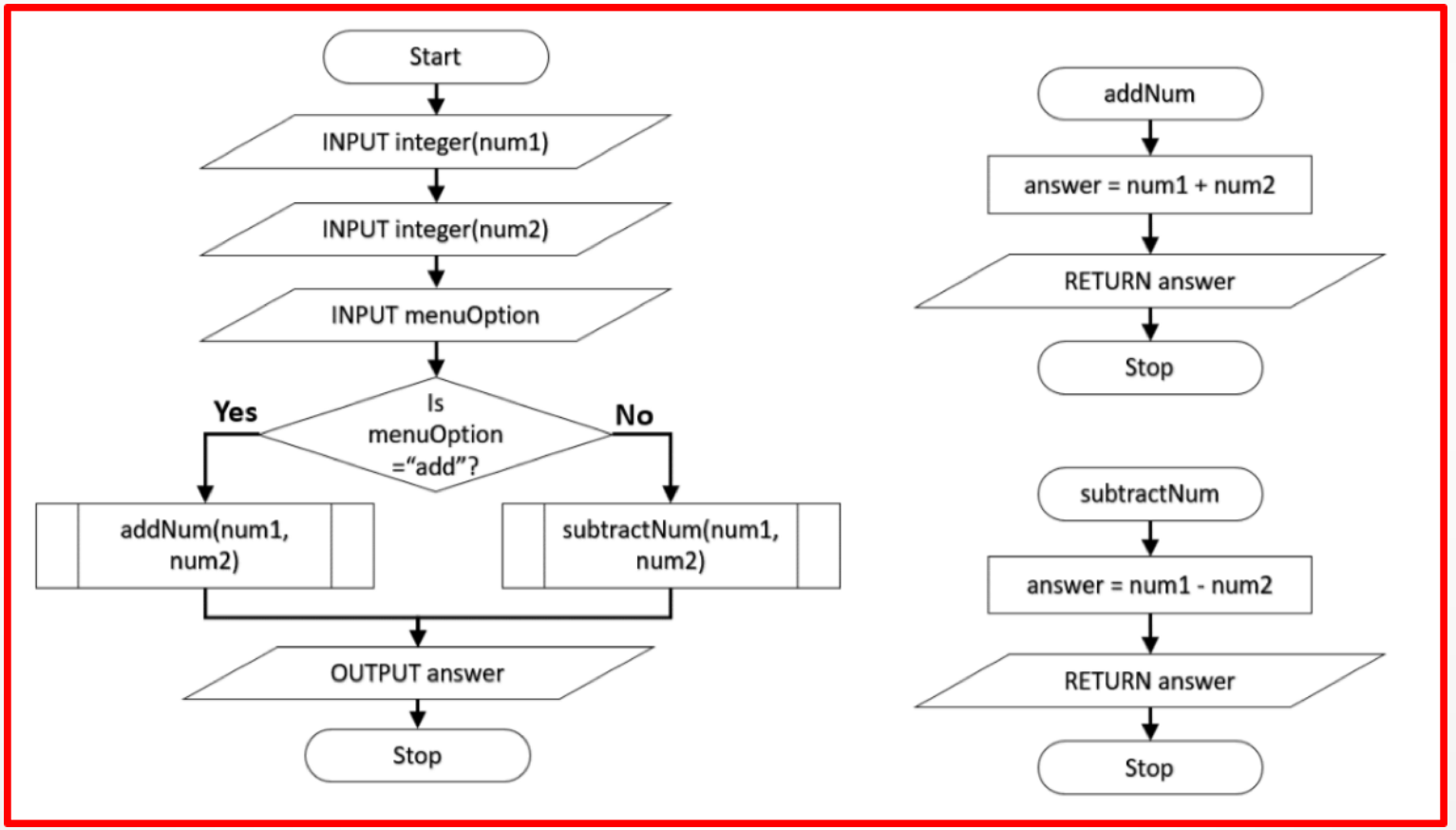
Στάδια μεταγλώττισης & σύνδεσης προγράμματος



Υποπρογράμματα (ρουτίνες)

Ακολουθία εντολών για εκτέλεση συγκεκριμένου task, «πακεταρισμένη» ως αυτόνομη μονάδα (https://en.wikipedia.org/wiki/Subroutine)

- Εξωτερικά (language or system libraries)
- Εντός προγράμματος (routine, subprogram, function, method, or procedure, κ.α.)



Εκσφαλμάτωση (Debugging)

Κατηγορίες σφαλμάτων

- Κατά την συγγραφή/μεταγλώττιση (compile errors)
  - συντακτικά, π.χ. ορθογραφικό λάθος σε εντολή
  - Σημασιολογικά π.χ. ασυμβατότητα τύπων μεταβλητών σε γλώσσα που επιβάλει έλεγχο τύπου

  Ο μεταγλωττιστής επισημαίνει το σημείο και το είδος του λάθους, ενδέχεται να προτείνει και σχετική διόρθωση.
- Κατά την εκτέλεση (runtime errors)
  - Συμβαίνουν προβληματικές καταστάσεις οι οποίες αναγκάζουν το διερμηνευτή και/ή το λειτουργικό σύστημα να παρέμβει διακόπτοντας απότομα την εκτέλεση του προγράμματος (crash, hanging). Ενδεικτικά:
    - - εγγενείς αιτίες, π.χ. διαίρεση με το μηδέν, αναφορά εκτός ορίων πίνακα, υπερχείλιση θέσης μνήμης μεταβλητής, κ.α.
    - Εξωτερικά – τυχαία αίτια, π.χ. βλάβη σκληρού δίσκου, διακοπή τηλεπικοινωνιακής σύνδεσης δικτύου, κ.α.
- Λογικά λάθη
  - Δεν ανιχνεύονται από το μεταγλωττιστή, ούτε διακόπτουν την εκτέλεση, αλλά παράγουν λανθασμένα αποτελέσματα, π.χ x=y*2 αντί x=y+2, if (a > 2), αντί if (a<2), κ.α.





Εκσφαλμάτωση (Debugging)

1. Εργαλεία εκσφαλμάτωσης (debuggers) που μας παρέχουν:

- εκφράσεις ελέγχου (watch expressions)
  - παρατηρούμε τη συμπεριφορά μιας μεταβλητής ή έκφρασης κατά τη διάρκεια εκτέλεσης του προγράμματος.
- σημείο διακοπής (breakpoint)
- Βήμα προς βήμα (step-by-step) εκτέλεση
- Ιστορικό (history)
- ιχνηλάτηση (tracing)
- άμεση εκτέλεση εντολής

2. Χωρίς debugger: Παρεμβάλω σε κρίσιμα σημεία του προγράμματος εντολών που εκτυπώνουν
  - Τιμές μεταβλητών ή εκφράσεων που με προβληματίζουν
  - Απλό μήνυμα (κείμενο) ώστε να καταλάβω ότι ο έλεγχος ροής προγράμματος πέρασε από το αντίστοιχο σημείο του κώδικα (όσες φορές βλέπω εκτύπωση)





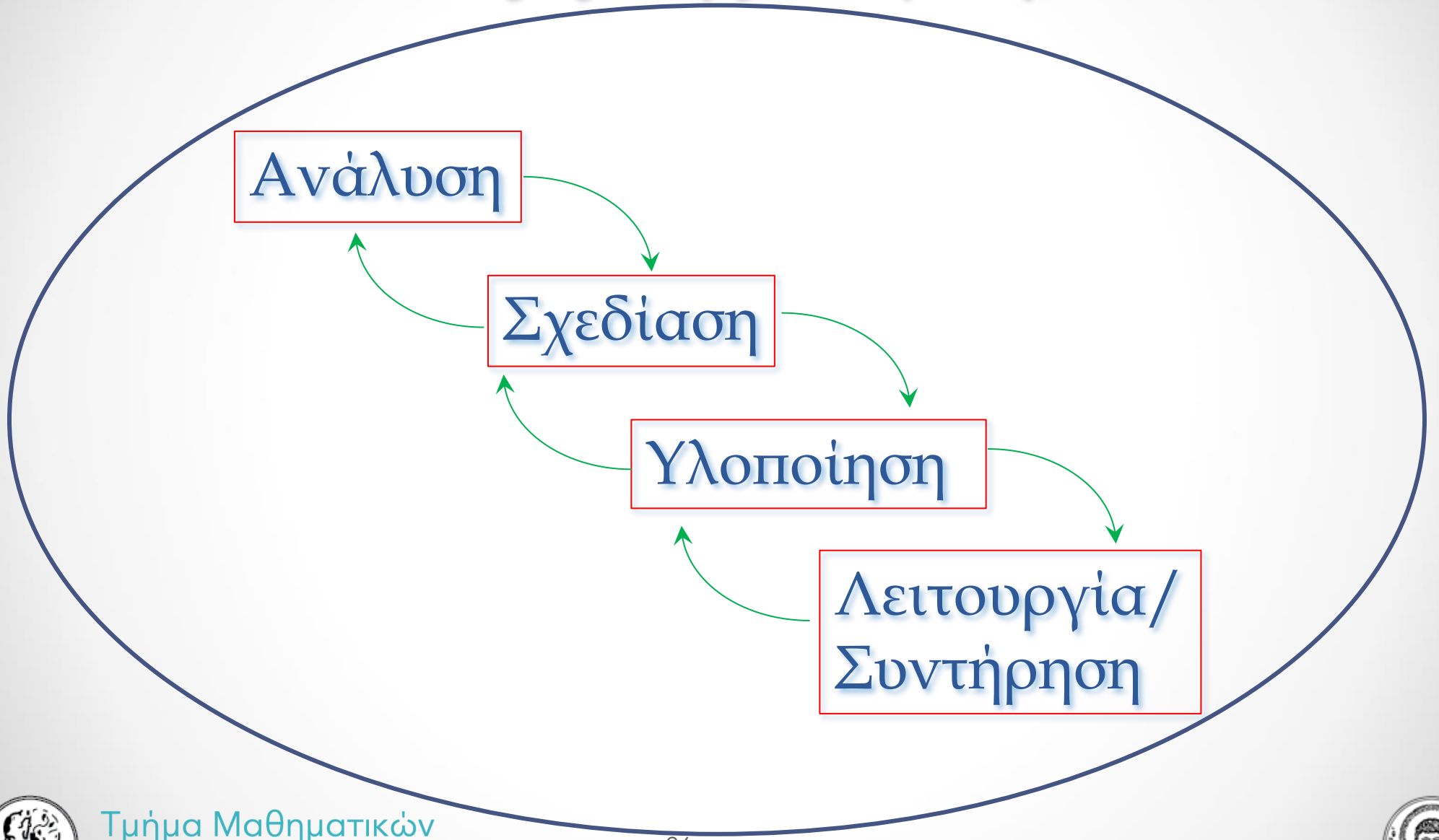
Κύκλος ζωής λογισμικού

- Ανάλυση
- Σχεδίαση
- Υλοποίηση
- Λειτουργία
- Συντήρηση (Αναβάθμιση, Επιδιόρθωση)





Κύκλος ζωής λογισμικού



Επιλογή γλώσσας προγραμματισμού

Η γλώσσα είναι το εργαλείο του προγραμματιστή, προκειμένου ο υπολογιστής να εκτελέσει μια εργασία.

- Ποιο εργαλείο διαλέγω για να γίνει η «δουλειά»;
- Εξαρτάται απ' τη δουλειά! **ΔΕΝ** υπάρχει μια και μοναδική «καλύτερη» γλώσσα προγραμματισμού!!!

Κριτήρια επιλογής γλώσσας προγραμματισμού:

- Είδος εφαρμογής
- Υπολογιστικό περιβάλλον που έχουμε στη διάθεση μας
- Διαθέσιμα προγραμματιστικά περιβάλλοντα (μεταφραστές κλπ.)
- Γνώσεις των προγραμματιστών (και προσωπικές προτιμήσεις)
- Προθεσμία παράδοσης
- Απαιτήσεις για βελτιστοποίηση πόρων
- Οικονομικό κόστος

Επιλογή γλώσσας προγραμματισμού

«Κλειστό» (proprietary) λογισμικό VS Ελεύθερο Λογισμικό / Λογισμικό Ανοικτού Κώδικα (ΕΛ/ΛΑΚ)

https://en.wikipedia.org/wiki/Open-source_software

https://en.wikipedia.org/wiki/The_Open_Source_Definition



1. Free Redistribution
2. Source Code
3. Derived Works
4. Integrity of The Author's Source Code
5. No Discrimination Against Persons or Groups
6. No Discrimination Against Fields of Endeavor
7. Distribution of License
8. License Must Not Be Specific to a Product.
9. License Must Not Restrict Other Software
10. License Must Be Technology-Neutral





Σημ: Ελεύθερο δεν σημαίνει άνευ κόστους (σε χρόνο, τεχνογνωσία, κλπ.)

Πρακτικές συμβουλές προγραμματισμού

1. Γράφουμε απλά και κατανοητά προγράμματά, ώστε ο (ίδιος ή άλλος) προγραμματιστής να μπορεί (πολύ) αργότερα να καταλάβει, τροποποιήσει ή/και επεκτείνει το πρόγραμμα.
2. Τα ονόματα σταθερών και μεταβλητών να υπονοούν τη χρήση τους.
3. Βάζουμε επεξηγηματικά σχόλια, ειδικά στα πιο δυσκολονόητα σημεία του προγράμματος.
4. Οι κενές γραμμών στα όρια των ενοτήτων του προγράμματος το κάνουν πιο ευανάγνωστο.
5. Η χρήση σταθερών διευκολύνει επόμενες αλλαγές αποτρέποντας ξεχασμένες παλιές τιμές είτε ακούσιες αλλαγές.
6. Θέτουμε αρχικές τιμές στις μεταβλητές του προγράμματος, ώστε η υλοποίηση να μην εξαρτάται αποκλειστικά από τυχόν αυτόματη αρχικοποίηση από το (κάθε) προγραμματιστικό περιβάλλον (μεταγλωττιστή).
7. Σπάμε μεγάλους υπολογισμούς σε απλούστερους για ευκολότερη κατανόηση του προγράμματος και αποφυγή λαθών.





Περισσότερες πρακτικές συμβουλές

1. Σε σύνθετες λογικές εκφράσεις προσέχουμε την ιεραρχία των τελεστών. Καλύτερα αχρείαστες παρενθέσεις, παρά λάθη, αβλεψίες και δυσανάγνωστος κώδικας…
2. Πριν βάλουμε φωλιασμένα (nested) IFs, ελέγχουμε μήπως ο ίδιος υπολογισμός μπορεί να υλοποιηθεί με σύνθετες λογικές εκφράσεις, ή IF-THEN-ELSE ή κάποια άλλη διαθέσιμη εντολή επιλογής της γλώσσας.
3. Μεταβλητές που ελέγχουν την επανάληψη βρόγχων τύπου WHILE, DO WHILE, REPEAT-UNTIL πρέπει να αλλάζουν τιμή μέσα στο σώμα του βρόχου, αλλιώς ο βρόγχος ουδέποτε εκτελείται ή (συνήθως) δεν τερματίζει η εκτέλεσή του (ατέρμων βρόχος – infinite loop).
4. Επανάληψη με την εντολή WHILE, μπορεί να μην εκτελεστεί ούτε μία φορά, αφού ο έλεγχος γίνεται στην είσοδο του βρόχου, ενώ με DO-WHILE θα τρέξει τουλάχιστον μία φορά.
5. Η εντολή FOR χρησιμοποιείται μόνο για προκαθορισμένο αριθμό επαναλήψεων, ο οποίος πρέπει να είναι ήδη καθορισμένος ή υπολογίσιμος.
6. Αποφύγουμε την αλλαγή της αρχικής/τελικής τιμής, του βήματος ή της μεταβλητής που ελέγχει την επανάληψη μέσα σε ένα βρόχο FOR, ακόμα κι αν η γλώσσα προγραμματισμού το επιτρέπει, διότι το πρόγραμμα γίνεται δυσνόητο και πιθανά λανθασμένο.





Πρακτικές συμβουλές για πίνακες

1. Χρειάζεται πραγματικά πίνακας για την επίλυση του προβλήματος; Αν δεν είναι απαραίτητος, δεν τον χρησιμοποιούμε (οι πίνακες ξοδεύουν μεγάλα ποσά μνήμης).
2. Αποφεύγουμε τα πλέον συνήθη λάθη στη χρήση των πινάκων, προσέχοντας πάντα:

   • Να δίνονται αρχικές τιμές σε όλους τους πίνακες.

   • Μην ξεπερνιούνται τα όρια του κάθε πίνακα. Το πιο συνηθισμένο λάθος στη χρήση των πινάκων είναι η προσπάθεια ανάγνωσης ή εκχώρησης τιμής έξω από τα όρια του πίνακα.

   • Η επεξεργασία γίνεται στα στοιχεία του πίνακα (ένα προς ένα). Συνεπώς σε όλες τις εντολές πρέπει να εμφανίζονται τα στοιχεία του πίνακα και όχι το όνομα του ίδιου του πίνακα.
3. Όλα τα στοιχεία του πίνακα έχουν τον ίδιο τύπο, π.χ. όλα είναι ακέραια ή όλα είναι χαρακτήρες όπως ορίστηκαν στο οικείο τμήμα δηλώσεων.
4. Επιλέγουμε και χρησιμοποιούμε την πλέον κατάλληλη μέθοδο για την ταξινόμηση ή την αναζήτηση σε έναν πίνακα.





Πρακτικές συμβουλές για υπορουτίνες

- Μελετούμε πως ένα πρόγραμμα μπορεί να αναλυθεί σε επιμέρους τμήματα και αποφασίζουμε για τα αντίστοιχα υποπρογράμματα **πριν** ξεκινήσουμε τη συγγραφή του προγράμματος. Η ιεραρχία των υποπρογραμμάτων σχεδιάζεται σε ένα διάγραμμα. Κατόπιν αναπτύσσουμε τον αλγόριθμο για κάθε υποπρόγραμμα και μετά γράφουμε το πρόγραμμα.
- Μελετούμε αν ένα υποπρόγραμμα είναι σκόπιμο να υλοποιηθεί με διαδικασία ή με συνάρτηση.
- Εξετάζουμε αν υπάρχουν έτοιμες βιβλιοθήκες προγραμμάτων οι οποίες μπορούν να χρησιμοποιηθούν αντί να γράψουμε από την αρχή κάποια υποπρογράμματα, ώστε να εξοικονομήσουμε χρόνο και κόπο.
- Προσπαθούμε να κρατήσουμε κάθε υποπρόγραμμα όσο είναι δυνατόν πιο ανεξάρτητο από τα άλλα. Έτσι αποφεύγουμε λάθη στο πρόγραμμά και μπορεί να ξαναχρησιμοποιήσουμε μελλοντικά το ίδιο υποπρόγραμμα σε άλλα προγράμματα.
- Προσέχουμε τα πλέον κοινά λάθη: Ορίζουμε τον τύπο της συνάρτησης. Οι συναρτήσεις παράγουν μόνο ένα αποτέλεσμα συγκεκριμένου τύπου, ακεραίου, πραγματικού κ.λπ. Ο οποίος πρέπει να ορίζεται.
- Προσέχουμε την σωστή αντιστοίχιση τυπικών και πραγματικών παραμέτρων.





Πρακτικές συμβουλές … γενικώς

- Keep It Simple (KIS)
- If it works*, don’t touch it! (* with an acceptable level of quality/efficiency)
- Read The Manual (RTM)
- Before you ask : Read the FAQ (Frequently Asked Questions) page





Ερωτήσεις & Απαντήσεις



